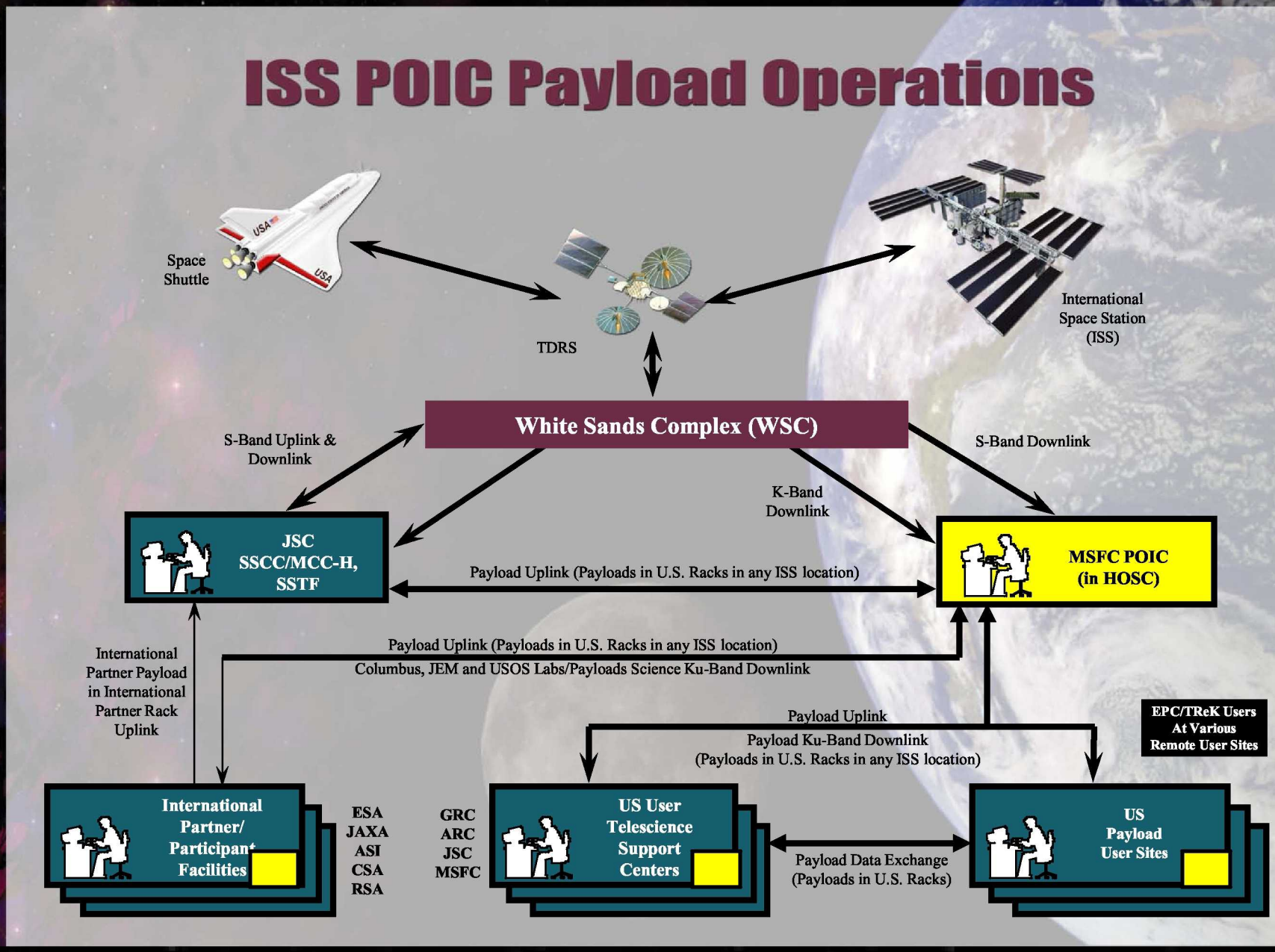
The Next Generation of Ground Operations Command and Control;

Scripting in C# and Visual Basic

George Ritter
Computer Science Corporation (CSC), MSFC, Huntsville, Alabama
And
Ramon Pedoto
COLSA Corporation, MSFC, Huntsville, Alabama

Scripting in C# and Visual Basic (at POIC)



Introduction

- Script: A pre-planned set of instructions to execute or perform a task such as the dialog in a play or a "small" job in a computer.
- Scripts have become mainstream tools for solving more of the "house-keeping" computer problems such as managing file systems
- Where compiled programming languages focus on performance, scripts are geared towards ease of use.
  - Ease of use lends itself to being interpreted
- MSFC has developed a tool called the Enhanced and Redesigned Scripting Language (ERS)
- ERS combines the ease of use of the typical interpreted scripting language with the power and richness of a full featured programming language.
  - Yup, ERS Scripts are complied
  - No, don't leave yet ☺

Scripts For Ground Ops

- From JCL to Ruby/Perl, Sys Admins and software developers have sought to "avoid the compiler" in an effort to simplify common and repetitive tasks that are not as performance oriented
- Space ground systems have adapted and extended scripting concepts to the command and telemetry processing domains in order to "simplify repetitive tasks."
- What makes the interpreted languages so useful in the Space Operations world?
- Are there other solutions that combines the graphical richness of a complied language with the ease and flexibility of a scripting development environment?

Scripting Mini-History

- 1960's: IBM introduces (JCL) on their OS/360 where files could be copied from one location to another using only 9 lines of instructions!
- JCL was followed by Data General's (CLI) and Unix Bourne Shell.
- These Scripts store a series of commands in a file. Data General called them "macros". Unix called them "shell scripts." In all cases, the scripts ran as interpreted statements, no compiling.
- Local variables and flow control were slowly added. As complexity grew, these languages continued to be interpreted, most likely because compute power was also increasing.
- Today's dynamic scripting languages, include Perl, Python, Ruby, Hypertext Preprocessor (PHP), Active Server Page (ASP), and JavaScript
- Today's scripting languages have progressed beyond simple file manipulation to complex programming tasks. They satisfy the needs of providing fast-to-develop and easy-to-maintain programming solutions in many of today's computer problem domains.

A Few Space Ground Ops Scripting Languages

- **SCL** by SRA International *"greatly reduces workload and automates routine tasks through procedural, time-sequenced and event based responses to real-time data."*

- *TSTOL, derived from System Test and Operations Language (STOL), is a procedural command language consisting of a core set of generic commands, supplemented by mission-specific extensions."*
  - TSTOL includes typical programming capabilities such as various data types, arithmetic, logical, and relational operators, global and local variables, and looping constructs.
  - TSTOL also has built in "procedures" or commands specific to the Goddard Mission Systems.

- **SLP (Scripting Language Processor)** at the MSFC ISS POIC, and also currently used by the Chandra X-ray Observatory Control Center, is based on the TSTOL design
  - In SLP many of the same things can be done as in TSTOL with respect to arithmetic, logical, and looping functions.
  - The SLP directives also include POIC/Chandra command and telemetry specific procedures for remote control of the ISS Payload and Chandra spacecraft.

Some Advantages of Scripts in Operations

- Scripts are easier to write due to the simple and limited language syntax.
- A single Scripts statement does lots of work, i.e., "uplink command,"
- The script developer gets immediate feedback from the interpreter.
- Scripts provide easy to use constructs for user-controlled program flow.
  - It is necessary to have ground operations personnel monitor the script. With SCL, TSTOL, and SLP, the script user has many control options.

Any Disadvantages of Scripts?

- Scripting languages are not historically known for being rich in graphic capabilities.
  - At the MSFC POIC, the operations Cadre personnel use a combination of scripts (SLP and now ERS), a data display tool, and custom programs or comps (compiled languages) to provide them with the best combination of all the features they need to automate their flight operations tasks.
- Why reinvent and maintain program flow constructs?
- Are there better options for debug environments?
- Do Scripting Languages really have a simple and limited syntax?

The Seed for ERS

- Is it possible to combine the features of a scripting language with the graphical and IDE richness of a programming language like Visual Basic.
- Some scripting language syntax had become more complex and more work to maintain just to provide features that basic programming languages already offer.
- Visual Basic and C# programmers are becoming more readily available.
- Let us allow programming languages do what they do best and we add in classes and methods (script commands or directives) that do lots of work, i.e. “uplink command.”
- But we have to develop a way to provide run-time “script” control that made execution feel and act like a script.
- Thus was born the Enhanced and Redesigned Scripting language or ERS.

ERS

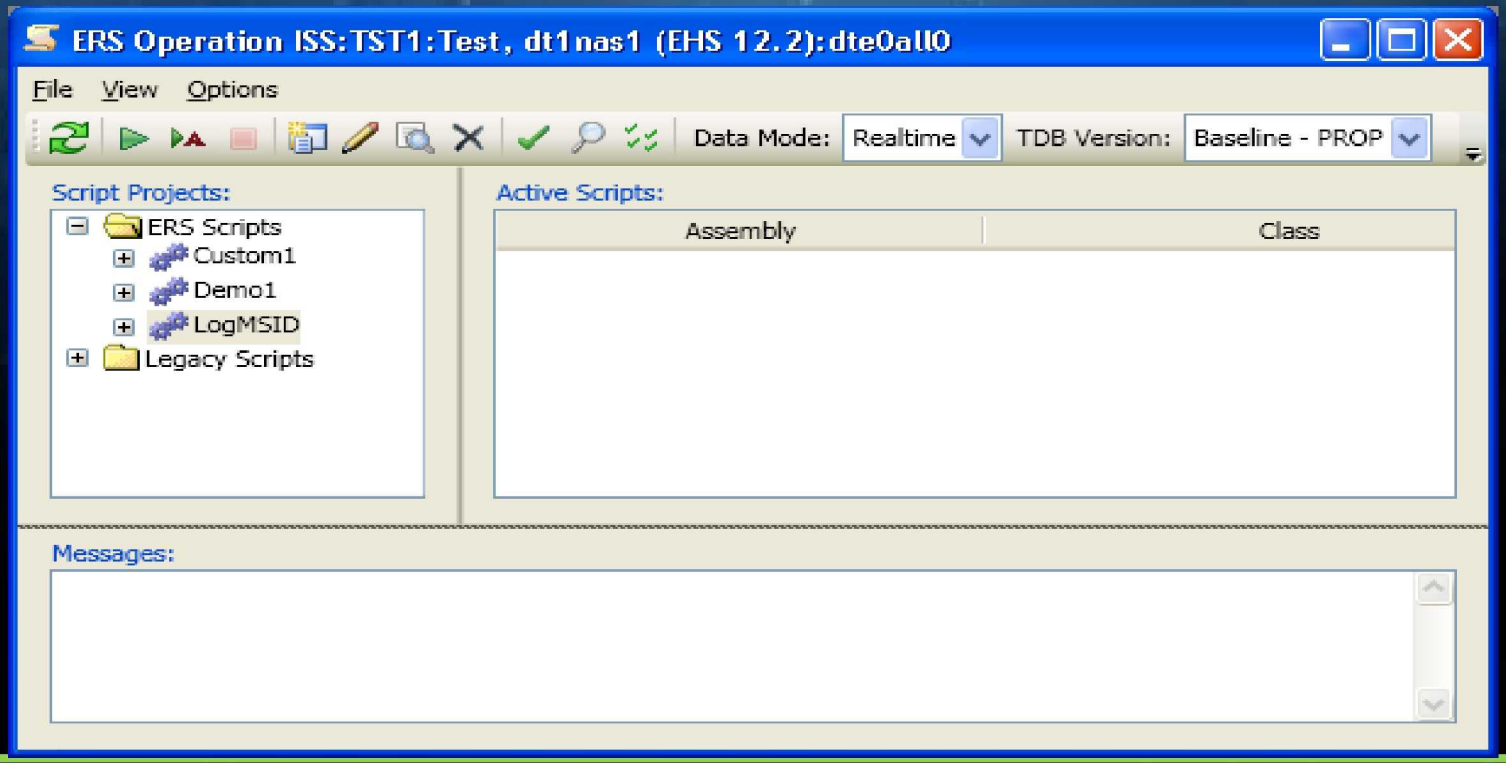
- ERS uses the Microsoft Visual Studio programming environment and offers custom controls that enable an ERS developer to extend the Visual Basic and Visual C# languages to interface with the POIC telemetry and command system

- ERS Offers:
  - the richness of a full featured programming language
  - Simplicity of integration with POIC command and telemetry services
  - execution control features that make an ERS program feel and operate like a script.

- ERS consists of 3 major pieces
  1. ERS Operations: creating, starting, stopping….
  2. Microsoft Visual Studio (can be free: Visual Studio Express)
  3. Custom wizards developed at Marshall to provide point and clink integration with the POIC ground system's core libraries.

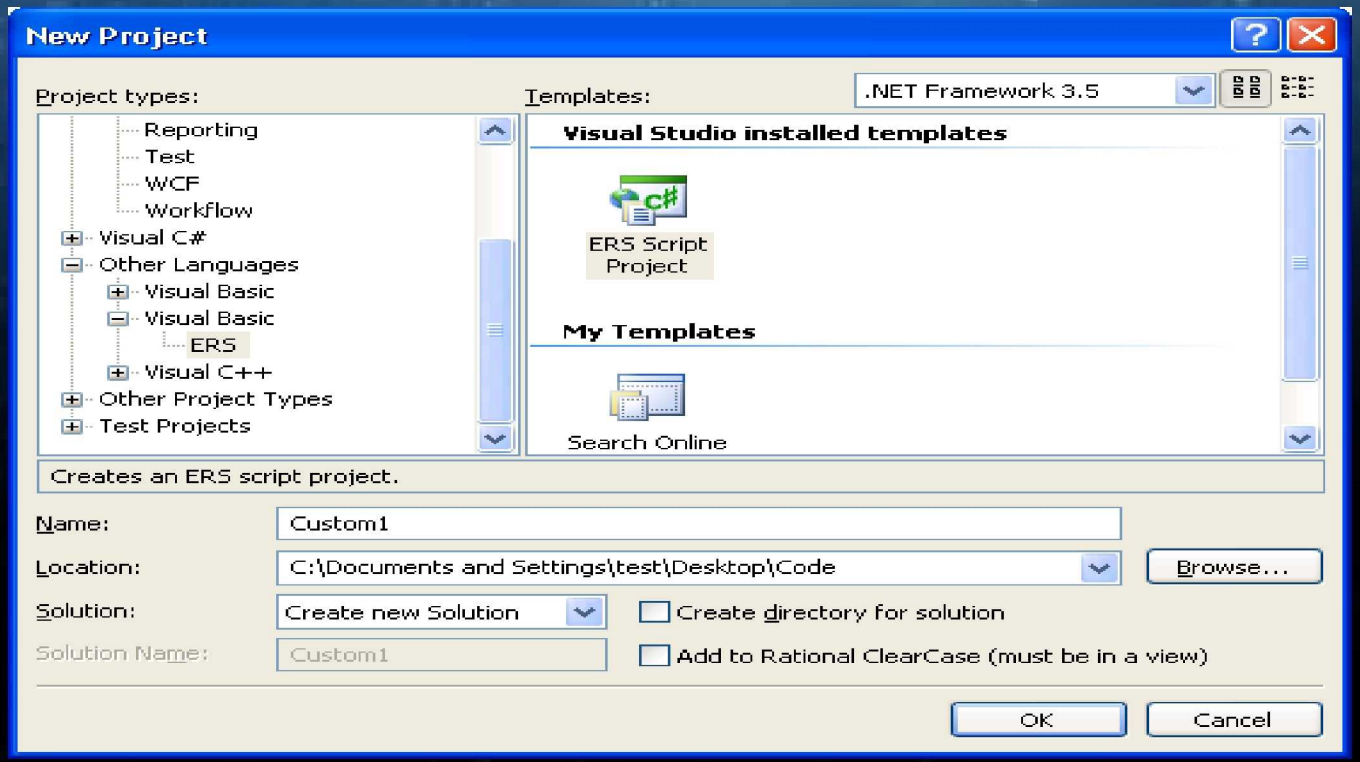
ERS Operation

- In ERS Operation the user can create, edit, delete, validate, and run ERS scripting projects.
- When creating a new ERS project, the ERS developer selects "add new script" from the File menu option in ERS Operation, which will launch Visual Studio's New Project Wizard



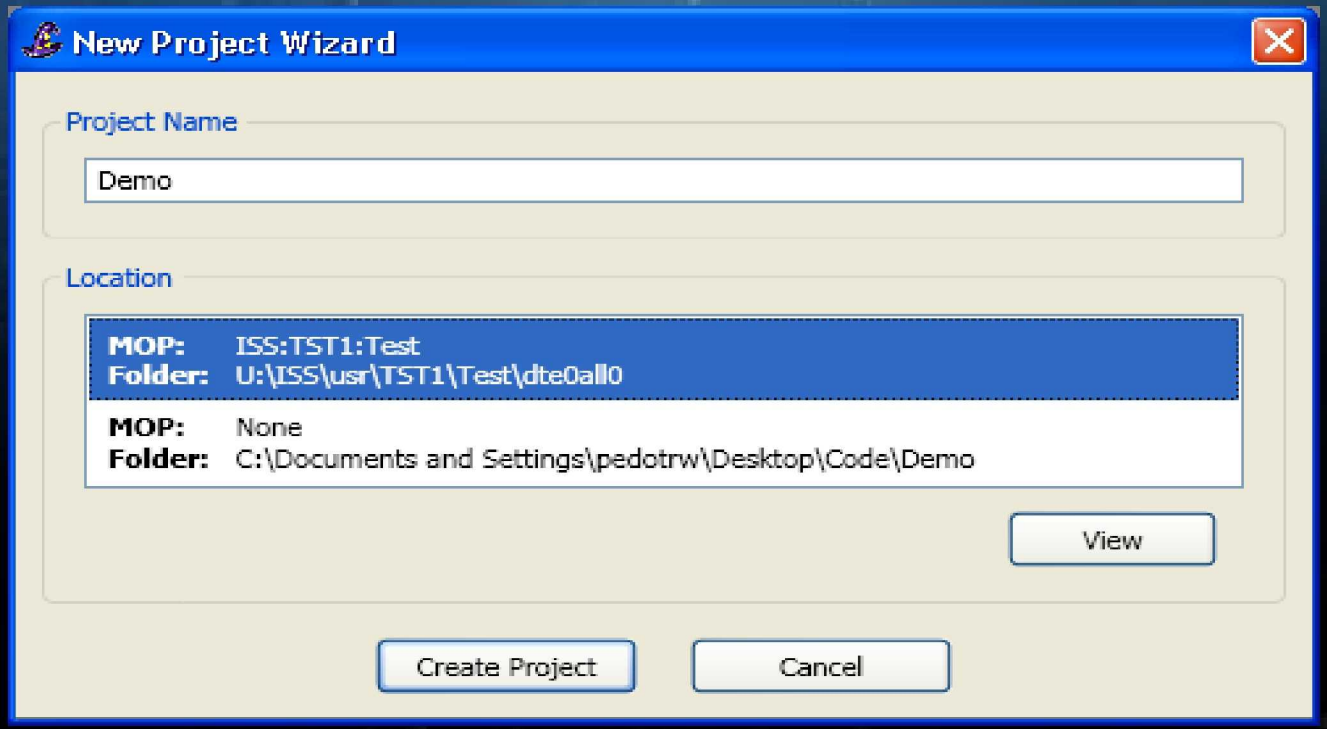
Visual Studio Project Wizard

- In the Visual Studio Project Wizard, the user then selects a desired programming language, the ERS Script Project template, and a project name.
- Clicking "okay" brings up the ERS New Project Wizard



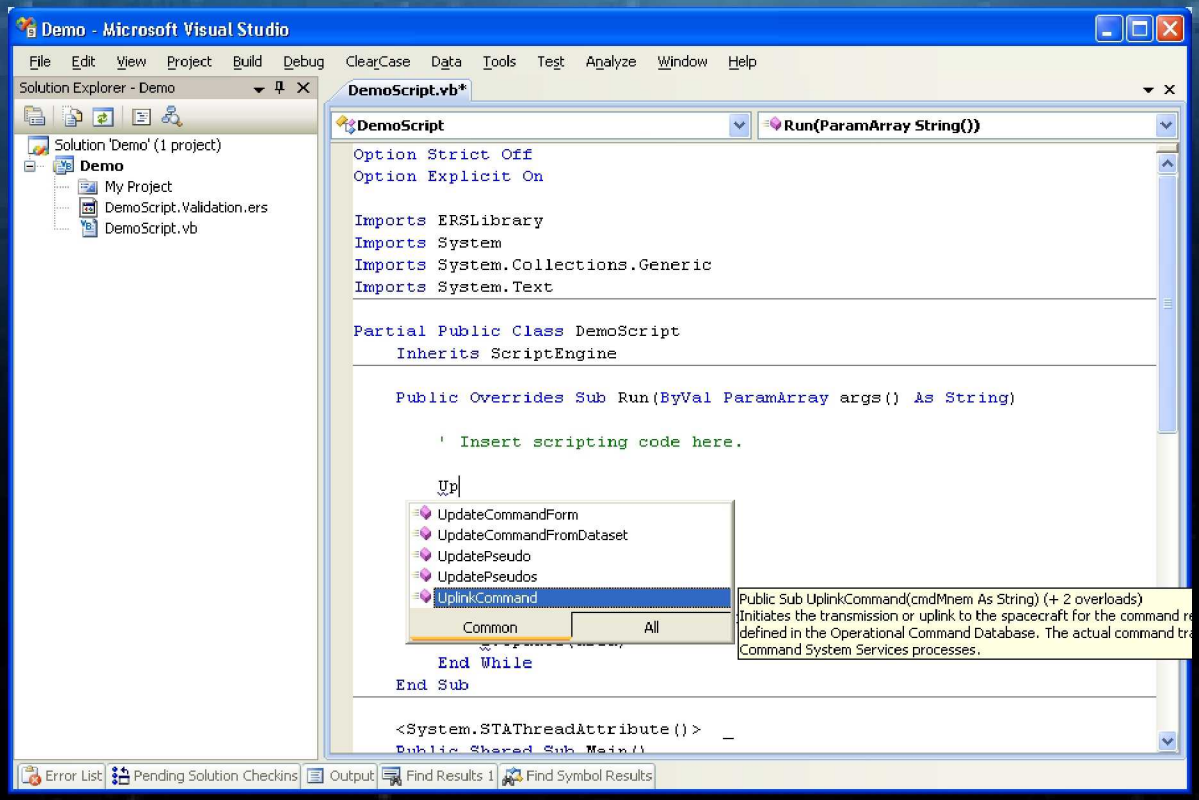
ERS Project Wizard

- Here the user must select an active "EPC session" to run against. The EPC session selection connects the ERS project to a specific set of telemetry and command meta-data within the POIC ground system.
- Clicking "okay" causes this Wizard to create a new .NET project that contains a code file plus an ERS-specific Validation File

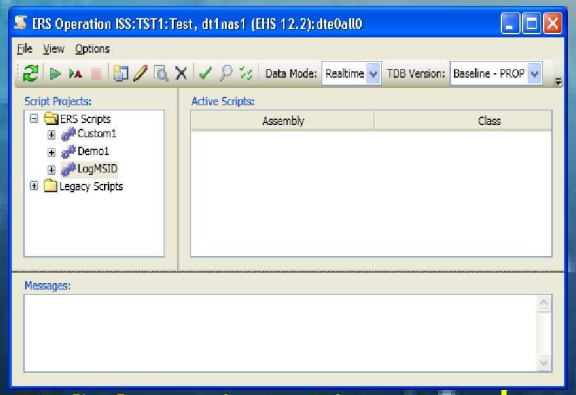


ERS Development

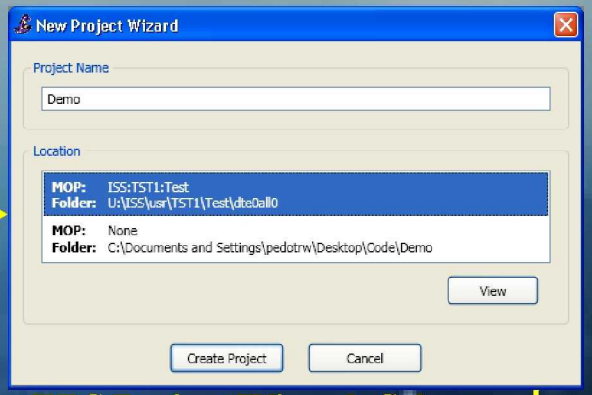
- An ERS script is a custom .NET class that extends a base class called "ScriptEngine".
- This base class is defined in the ERS library and contains many useful "EPC/POIC methods."
- These methods will appear in Visual Studio's Intellisense, whenever the user types code.
- Included in the Intellisense are descriptions for each method.
- User code goes after the green "insert scripting code here"



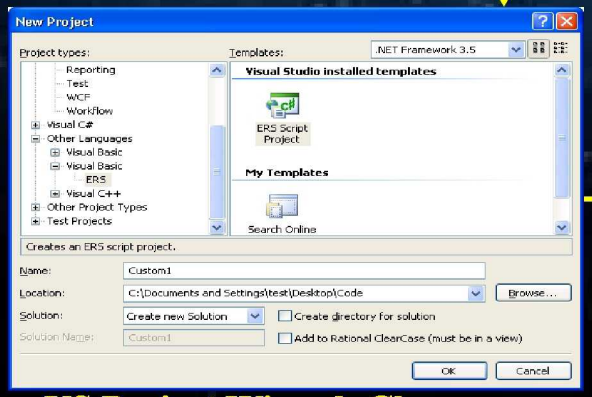
ERS Re-Cap



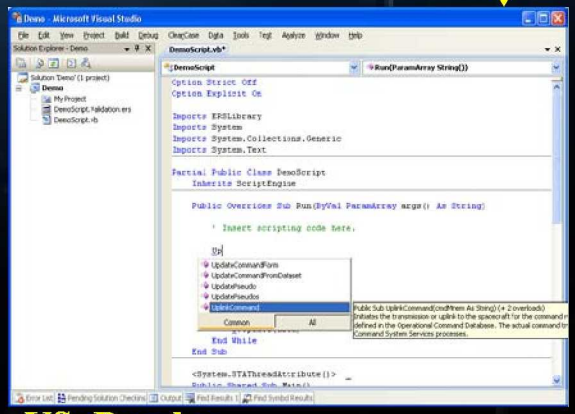
ERS Operation > File >New



ERS Project Wizard: Select TLM and CMD Meta Data



VS Project Wizard: Choose Language, Template, Name



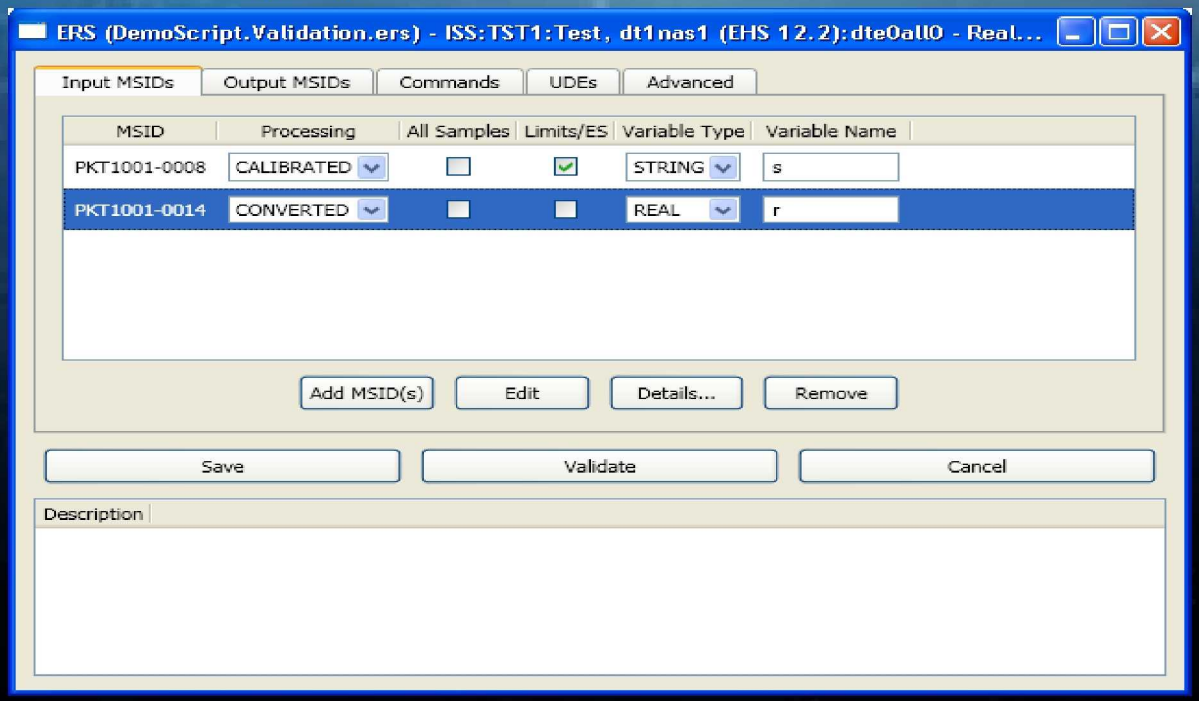
VS: Develop

Sample ERS "Directives

- Ask (variable): Prompts the operator for input and waits until a resume directive is entered. The value entered is stored in variable.
- Sample Latest (): Updates all MSIDs (variables) with their latest packet values.
- Sample Next (): Updates all MSIDs (variables) with their next packet values.
- Uplink Command (String, Verify): Initiates the transmission or uplink to the spacecraft for the command referenced by a unique mnemonic (variable) as defined by the Meta Data selection.

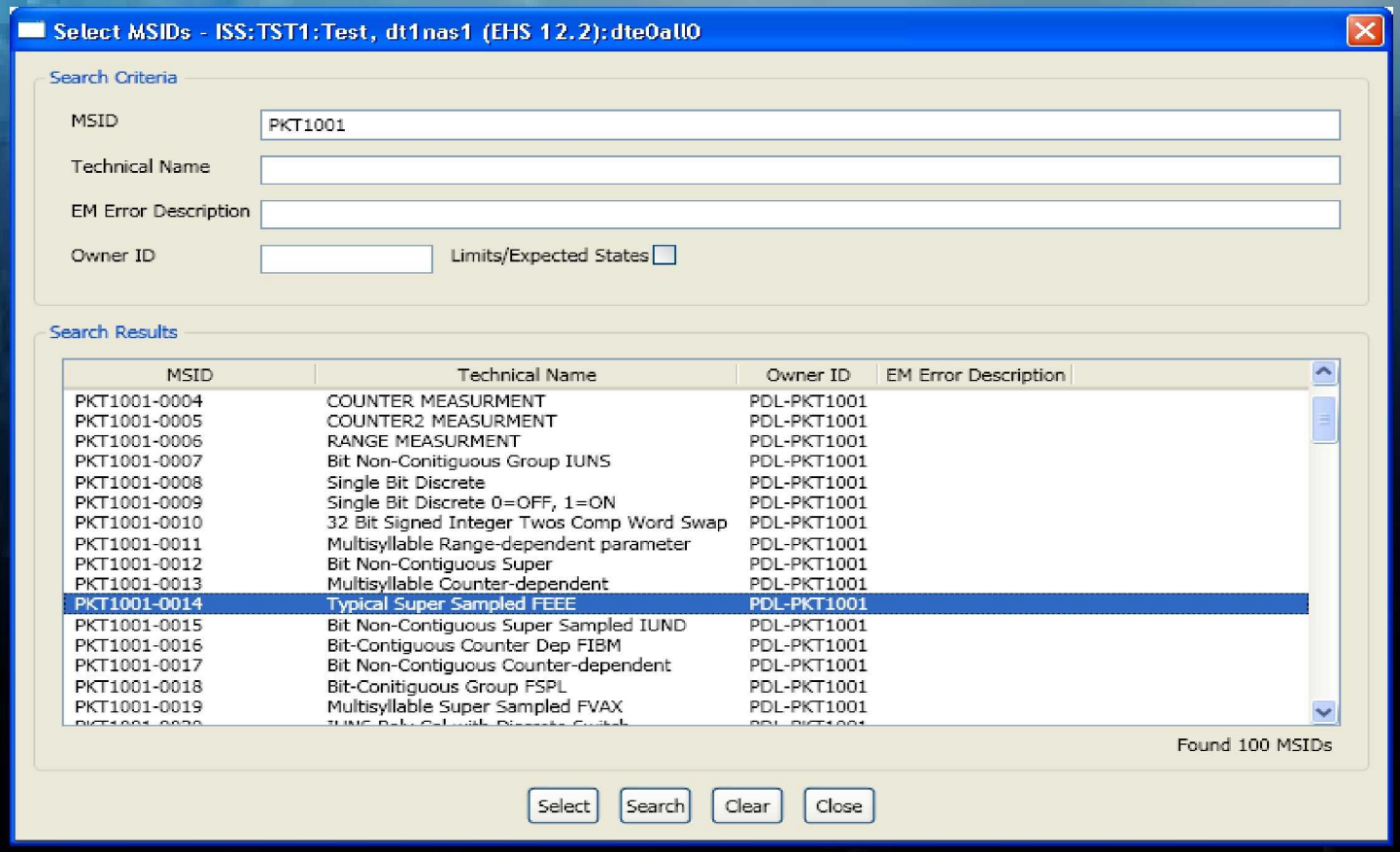
Telemetry and Commands as Variables

- To use the methods or procedures previously presented, individual telemetry mnemonics and Command Mnemonics must be assigned to variables in the program.




```
If STRING.IsDataNew Then
        Write("New data is available!")
End If
```

Choosing a Telemetry/CMD Mnemonic



Sample ERS Script

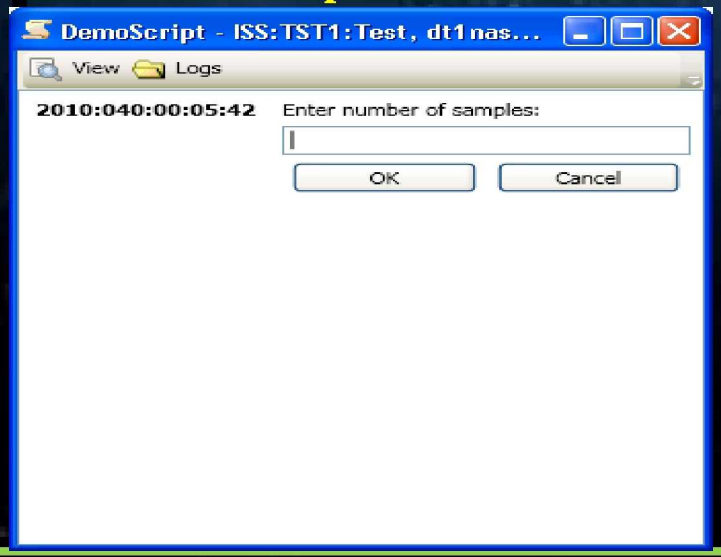
```
'Output a sample value 10 times
        For i = 1 To 10
            ' Get next packet of valid sample values
            SampleNextValid()
            ' Scale the first sample value of the MSID 'Pkt1001test'
            Dim scaledValue = Pkt1001test.Sample(0) * scale
            ' Write the scaled value back out as the Pseudo MSID 'Ext0test001'
            Ext0test001.Update(scaledValue)
        Next

' Set the modifiable fields on a command, update it in the DB, then uplink to spacecraft
        With AddDwnlinkFile.Fields.FileNames = "test.txt"
        End With
        ' Update the command in the database
        AddDwnlinkFile.Update()
        ' Verify that the user wants to uplink the command
        If AskPushButton("Are you sure you want to uplink?", "Yes", "No") = 0 Then
            ' Uplink the command
            AddDwnlinkFile.Uplink()
        End If
```

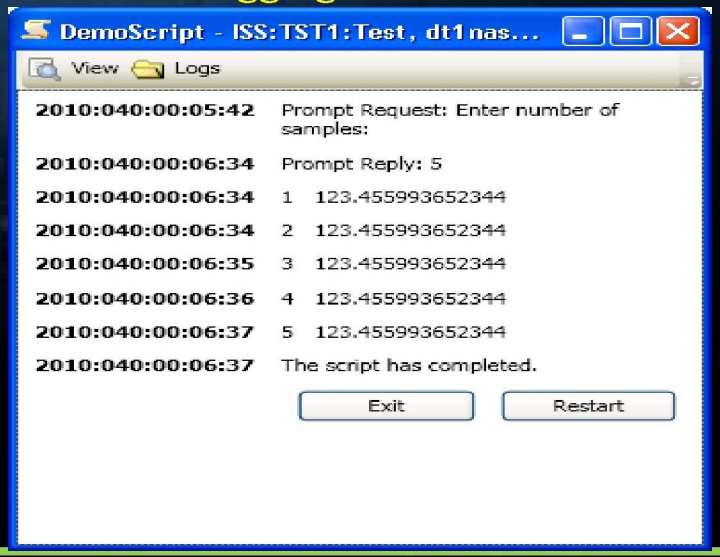
ERS Script Execution and Control

- ERS scripts can be controlled in debug mode inside the Visual Studio debugger where the script developer is provided with a rich set of the latest source level debug tools provided by Microsoft.
- ERS provides script control for a verified ERS script at run-time through ERS-provided input/output methods.
- These include methods to write out messages and to prompt a user for input.
- An ERS script that performs input and output is assigned its own logging window where both the input prompts and output messages are displayed.

User Input Window



Logging Window



Pre-Conclusion

- Scripts are a power tool for space ground operations offering a relatively simple syntax and an environment that is integrated well with the host ground system.
- Scripts offer flow-control functions that make them ideal for remote control including things like execution confirmation.
- However, scripting languages can be graphically challenged, and maintaining the typical program constructs becomes time consuming and unnecessary.

Conclusion

- ERS is Integrated with Microsoft's Visual Studio.
  - Lots of VB and C# coders, and we concentrate on adding more hooks to our ground system
- ERS wizards allow easy program startup and access to telemetry and command objects.
- ERS provides a powerful input/output flow-control feature that allows the user to monitor progress and for confirmation execution of critical tasks if necessary,
- ERS records of all user input/output actions in textual log files.
- ERS programs are not scripts, but ERS programs offer the same features as many scripting languages, while also having the ability to include rich graphical components.
- ERS is an Enhanced and Redesigned Scripting environment that is lowering our scripting development and maintenance costs while increasing our ground systems automation capability.

Nomenclature

ASP = Active Server Page
CLI = Command Line Interface
CSC = Computer Science Corporation
C2 = Command and Control
C3ISR = Command, Control, Communications, Intelligence, Surveillance and Reconnaissance
C# = 'C' "sharp", a programming language
EHS = Enhanced HOSC System
EPC = Enhanced Personal Computer (MSFC HOSC Telemetry and command tool-set)
ERS = Enhanced and Redesigned Scripting Language
HOSC = Huntsville Operations Support Center
HTML = Hypertext Markup Language
ISS = International Space Station
MSFC = Marshall Spaceflight Center
PHP = Hypertext Preprocessor
POIC = Payload Operations Integration Center
SCL = Systems Control Language
SLP = Scripting Language Processor
STOL = Systems Test and Operations Language
TSTOL = The System Test and Operations Language
VB = Visual Basic
VS = Visual Studio